Defense Technical Information Center

Compilation Part Notice

ADP023717

TITLE: On Software Protection in Embedded Systems

DISTRIBUTION: Approved for public release, distribution unlimited

This paper is part of the following report:

TITLE: Proceedings of the ARO Planning Workshop on Embedded Systems and Network Security Held in Raleigh, North Carolina on February 22-23, 2007

To order the complete compilation report, use: ADA485570

The component part is provided here to allow users access to individually authored sections of proceedings, annals, symposia, etc. However, the component should be considered within the context of the overall compilation report and not as a stand-alone technical report.

The following component part numbers comprise the compilation report:
ADP023711 thru ADP023727

On Software Protection in Embedded Systems

Jisoo Yang and Kang G. Shin

Department of Electrical Engineering and Computer Science

The University of Michigan

{jisooy,kgshin}@eecs.umich.edu

Abstract

We argue that the conventional privilege separation of a processor has inherent limitations in protecting software with higher security requirements, and hence, a new system of protection should be devised to overcome these limitations. To enable the new protection, an operating system needs to be restructured into two layers: the security kernel which implements the new protection system, and the management kernel which manages resources. The security kernel protects the applications even when the management kernel is compromised. The security kernel should be made very thin and simple, thus making it suitable for small devices like handsets and smart sensors & actuators.

Limitations of Current Software Protection

With increasing computation power and storage capacity, many embedded systems are adopting the paradigm of user/kernel separation of a processor [3] to provide better software protection and management. This protection paradigm is characterized by a complete separation of privilege. Code executing in user mode (i.e., applications) is prevented from performing sensitive operations, whereas code executing in kernel mode (i.e., operating system) is considered privileged and hence, given unlimited power.

This simple protection mechanism is effective in protecting the operating system (OS), but it does not serve well the security needs of user applications. In an embedded environment, where applications usually perform critical operations and carry sensitive data, the application must be protected as strongly as the OS. Although the OS provides certain protection to the applications, there are inherent limitations with the simple user/kernel separation and complete reliance on the OS for the application protection.

First, there is no effective second-line of defense that applications can resort to in case the OS is compromised.

Many applications need to protect secret/confidential information, but once an attacker seizes the control of the OS, it is very easy for the attacker to observe/steal the information. Any effort to further protect the information will be futile as the attacker can exploit the OS's power to subvert, reverse-engineer, or simply disable the protection mechanism.

Second, verifying the correctness of an OS is becoming intractable as its size and functionality continuously grow—even in an embedded environment—to meet the increasing demand for more features. Today's mobile phones, for instance, require some features comparable to those of PCs. Unfortunately, it is difficult to reduce the growing OS verification need due to the coarse-grained user/kernel separation, where every privileged code has unlimited power and thus, is subject to verification.

Third, the user/kernel separation generates trust dependencies among software components, which do not generally correspond to the relations of the component providers. This mismatch incurs assurance overhead and generates unwarranted conflicts of interest. For example, user applications must trust the OS. To trust the OS, the application providers need assurance of the OS's trustworthiness. For the assurance, a complete and unbiased validation of the OS is necessary, but doing so generally goes against the interest of the OS provider due to the cost of validation and the risk of exposing the system to others.

Challenges in Designing New Application Protection

We argue for the need of another system of protection that can deal with the above limitations. The new system should be able to protect applications even in the case of OS compromises, reduce the size of code required for verification, and break the trust dependencies between software components. To design and implement such a

1

protection system, we must overcome the following challenges.

The first challenge is to define an appropriate threat model for user applications. We need to identify the security properties that the applications/users want for protecting their information/data, so that the new protection mechanism can preserve themselves even if the OS were compromised. Unfortunately, our problem is not in the secure communication domain, and thus, it is difficult to borrow familiar security properties from that domain. Also, we need to avoid over-protection for simplicity. Therefore, we need a threat model that represents the problem domain and captures essential security needs of the applications.

The second challenge is to preserve the OS's usual management power. With the new protection, however, the OS is restricted somewhat; the new protection system enforces certain rules and the OS is prevented from performing actions against the rules. However, the restriction should not obstruct the OS from performing a legitimate management job.

The third challenge is to find an implementation that is small and simple to verify. With the new protection, the OS can be verified less stringently, since applications can still be protected even when the OS fails (as a result of its compromise). However, the mechanism that implements the new protection should be fully trusted, and hence, the correctness of the implemented protection is critical to the security of the entire system.

Security Kernel

The new protection requires a different OS arrangement which consists of two layers of kernel: security and management kernels. Running with complete privilege, the security kernel is a very thin layer that only implements the new protection system. It must be fully trusted and must thus be rigorously verified. The management kernel, responsible for resource management and scheduling, is a restricted version of a conventional OS. As it runs on top of the security kernel, applications are still protected from any compromise in the management kernel. In this sense, it does not have to be trusted and verified. Both security and management kernels are protected from user applications by the traditional user/kernel separation.

Since it is small and simple enough, the security kernel can be realized entirely with hardware. Equipped with a circuitry that implements the protection logic, a processor can extend its ISA to expose a programming interface for the management kernel and user applications.

A software-only solution is also possible by using virtualization techniques. A virtual machine monitor (VMM) is capable of not only running multiple OSes,

but also realizing hardware extensions or implementing system services without actually changing the real machine [1]. The VMM is more privileged than the OS and its perimeter is safe. Therefore, the security kernel can be implemented inside of the VMM.

Although we can implement the security kernel by modifying a full-fledged VMM such as Xen [2], a full VMM is not necessary as we do not have to run multiple OSes. Instead, a lightweight security kernel is preferred only by using the techniques and constructs required to enable the hardware extensions and to safeguard the VMM's perimeter.

Impact and Outlook

The new software protection system will make long-term impacts since it relaxes many assumptions currently made when software systems are composed. For instance, the management OS is no longer assumed to be trusted, thus creating opportunities for design of ambitious distributed systems which were risky under the assumption of trusted OS. Also, existing software systems can be retroactively redesigned to exploit the enlarged design space, thus making them more reliable with minimal additional effort.

Conclusion

The conventional user/kernel separation is not sufficient to meet the growing demand for software protection in embedded systems. We argue for the need of a new protection mechanism that can protect user applications, lessen verification overhead, and break trust dependencies. The new protection is enforced by a 'security kernel' which can be realized as a lightweight software layer using virtualization techniques, making it suitable for small devices and embedded systems, such as handsets and smart sensors & actuators.

References

[1] Peter M. Chen and Brian D. Noble. When virtual is better than real. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, May 2001.

[2] Boris Dragovic, Keir Fraser, Steve Hand, Tim Harris, Alex Ho, Ian Pratt, Andrew Warfield, Paul Barham, and Rolf Neugebauer. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, Oct 2003.

[3] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sep 1975.

New Direction for Software Protection in Embedded Systems

Kang G. Shin

Department of EECS
University of Michigan
Feb 22, 2007

Background

- Why application software protection in distributed embedded systems?
  - In embedded systems, application programs perform mission-critical tasks and carry sensitive info
  - Privacy/integrity of these applications is critical to the security and robustness of any distributed embedded system
- Current approach: have the OS protect the applications
  - E.g., OS provides process isolation, crypto services, etc.
  - Apps must trust OS. OS should be protected by hardware
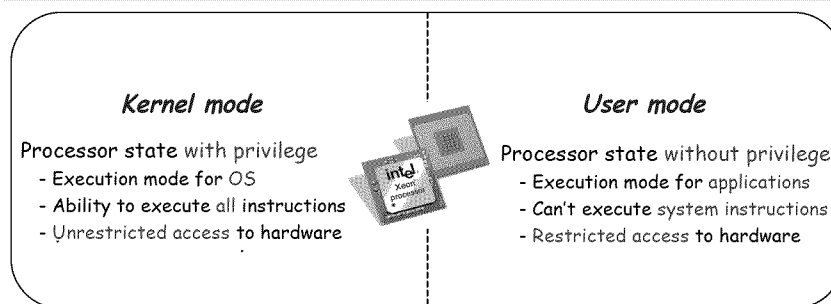 - Processor protects OS via user/kernel separation

2

My Position

Classical user/kernel separation is too coarse to confidently protect app software with high security needs. The limitation should be overcome by creating a new protection system.

3

Classical User/Kernel Separation

Kernel mode		User mode
Processor state with privilege		Processor state without privilege
- Execution mode for OS		- Execution mode for applications
- Ability to execute all instructions		- Can't execute system instructions
- Unrestricted access to hardware		- Restricted access to hardware

- An autocratic model for separation of power: the kernel code executes with absolute power
- Entire security of the system hinges on the trustworthiness of the kernel mode software (i.e., OS)
- Effective for protecting OS, but *this simple dichotomy is too coarse and there are several limitations*

4

Limitations of user/kernel Dichotomy

1. No defense against OS compromise
   - There is no effective 2nd line of defense to applications
   - Further protection of applications is meaningless as the attacker can easily disarm any protection mechanism
2. Difficult to reduce the OS verification overhead
   - Trend: OS is becoming larger and is from diverse sources
   - The dichotomy dictates any code that requires even slightest privilege must execute in kernel mode, where the code is subject to complete verification
3. Undue trust dependencies
   - App vendors require OS vendors not to spy on the apps
   - Apps must trust every component of OS
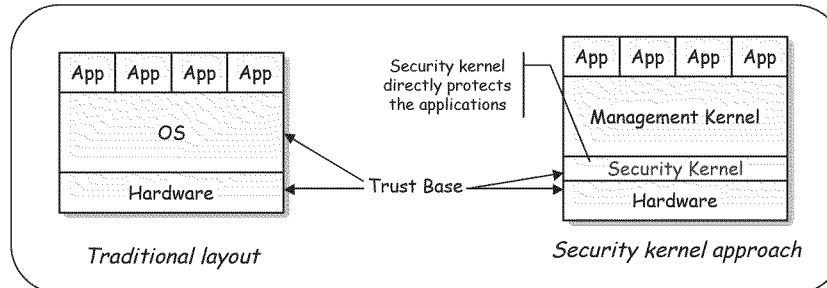   - Every OS component must be validated (e.g., device drivers)

5

New Directions for Software Protection

- Need a new protection system
  - Protect applications even in case of OS compromises
  - Lessen the kernel verification overhead
  - Break trust dependencies
- Challenges in designing such a protection system
  - Identifying an appropriate threat model
    - Model that captures essential security needs of apps
  - Preserving OS's management power
    - Restriction by the new protection shouldn't obstruct OS's job
  - Finding a small and simple enforcing mechanism
    - Implementation must be easily verifiable
- My proposal: Separate security from management
  - The new protection system protects privacy/integrity of apps.
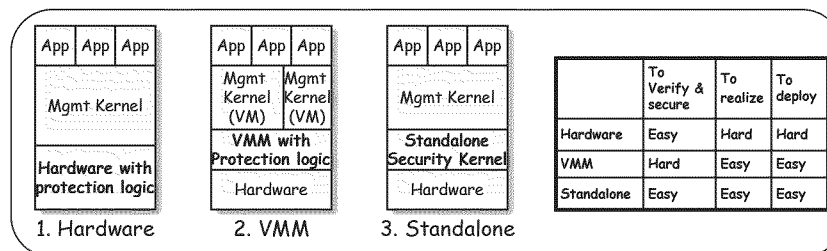 - It is implemented by a 'security kernel' (continue)

6

3

Security Kernel vs. Mgmt Kernel

App	App	App	App		App	App	App	App

Security kernel directly protects the applications

OS
Management Kernel
Security Kernel
Hardware — Trust Base — Hardware

Traditional layout   *Security kernel approach*

- Traditional OS → Security kernel + Management kernel
- Management kernel is responsible for resource management
- Security kernel is a thin layer enforcing the new protection system
  - It directly protects privacy/integrity of applications data
  - Applications are protected even if the management kernel is compromised
- Management kernel doesn't have to be trusted by applications

7

Implementation Alternatives

App App App

Mgmt Kernel

Hardware with protection logic

1. Hardware

App App App

Mgmt Kernel (VM) Mgmt Kernel (VM)

VMM with Protection logic

Hardware

2. VMM

App App App

Mgmt Kernel

Standalone Security Kernel

Hardware

3. Standalone

	To Verify & secure	To realize	To deploy
Hardware	Easy	Hard	Hard
VMM	Hard	Easy	Easy
Standalone	Easy	Easy	Easy

1. Hardware
   - Processor is modified to implement the protection logic
2. Software: Using virtual machine monitor (VMM)
   - A VMM, sitting between HW and OS, can be utilized
   - Due to size/complexity, verifying the VMM is challenging
3. Software: Standalone security kernel
   - A thin layer implementing only the protection system
   - Can be made small and simple, thus easy to secure

8

4

Impact

- Paradigm shift in designing secure distributed embedded systems
  - The new application protection system relaxes many assumptions currently made
  - To ensure the security of application software, management OS no longer has to be trusted
  - It enables implementing ambitious distributed systems which were too risky under the assumption of trusted OS

9

Conclusion

- Another system of protection is needed to overcome the limitations inherent with the coarse classical user/kernel separation.
- The protection system must
  - Protect applications even in case of OS compromises
  - Lessen the kernel verification overhead
  - Break trust dependencies
- We have been exploring approaches for implementing such a protection system

10